APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

VIRTUAL DIRECT MEMORY ACCESS CROSSOVER

Inventor(s): Daniel P. Baumberger

Prepared by: Justin B. Scout,
Reg. No. 54, 431

intel®
Intel Corporation

"Express Mail" label number: EV325530033US

VIRTUAL DIRECT MEMORY ACCESS CROSSOVER

BACKGROUND

1. Field

The present disclosure relates to the resource management of virtual machine(s) using information regarding the activity of the virtual machine(s), and, more specifically, to the facilitating communication between two or more virtual machines via the mapping of virtual resources to physical resources.

2. Background Information

The virtualization of machine resources has been of significant interest for some time; however, with processors becoming more diverse and complex, such as processors that are deeply pipelined/super pipelined, hyperthreaded, and processors having Explicitly Parallel Instruction Computing (EPIC) architecture, and with larger instruction and data caches, virtualization of machine resources is becoming an even greater interest.

Many attempts have been made to make virtualization more efficient. For example, some vendors offer software products that have a virtual machine system that permits a machine to be partitioned, such that the underlying hardware of the machine appears as one or more independently operating virtual machines (VM). Typically, a Virtual Machine Monitor (VMM) may be a thin layer of software running on a computer and presenting to other software an abstraction of one or more VMs. Each VM, on the other hand, may function as a self-contained platform, running its own operating system (OS), or a copy of the OS, and/or a software application. Software executing within a VM is collectively referred to as "guest software".

A typical VMM, which may be considered the host of the VMs, may enhance performance of a VM by permitting access to the underlying physical machine in some situations. A VM may see and interact with a set of virtual resources. These virtual resources may then be mapped to physical resources. The VMM is typically responsible for mapping the virtual devices to the physical devices. In general, the mapping process is the same for all devices regardless of how the device is used.

FIG. 1 is a block illustrating an embodiment of an apparatus 101 for mapping resources amongst two virtual machines 111 & 112 in accordance with a known technique. Both virtual machines see instances of virtual devices 131 & 132. In one example, the virtual devices may be network interface cards. These virtual devices may be mapped, by the VMM 180, to the physical device 190. In the example, the physical device may also be a network interface card.

Typically, when the first VM 111 wishes to use the virtual device 131, it may write data to the device's virtual memory element 121. The VMM 180 would assure that when the first VM attempts to write data to the virtual memory element, it is in fact written to the physical memory element 181 of the physical device 190. The same process occurs when the second VM 112 uses its corresponding virtual device 132. In FIG. 1, the first VM illustrates the write case, while the second VM illustrates the read case. However, bi-directional communication is often the norm.

In an example where the deices are network interfaces, if the first VM 111 wishes to communicate with a device outside the apparatus 101, the first VM would write information to the physical device 190, and the physical device would then transmit the data outside the apparatus. The second VM 112 illustrates the read case. In this

example, physical memory 181 illustrates the transmit buffer of the network interface, and physical memory 182 illustrates the receive buffer.

Typically, if the first VM 111 wishes to communicate with the second VM 112, the first VM attempts to communicate with the second VM as if the second VM was a device outside of the apparatus 101. As a result, data is written to the physical device 190, and then read from the physical device by the second virtual machine. In the network interface example, the loopback feature of the network interface is used to route the data from the first VM to the second VM, and vice versa.

This loopback solution is typically used by VMware, Inc. (VMware) of Palo Alto, California. The VMware solution provides for running two or more operating systems, such as Linux and Microsoft Windows, on a single machine, using the facilities provided by the operating system that runs on the underlying hardware.

BRIEF DESCRIPTION OF THE DRAWINGS

Subject matter is particularly pointed out and distinctly claimed in the concluding portions of the specification. The disclosed subject matter, however, both as to organization and the method of operation, together with objects, features and advantages thereof, may be best understood by a reference to the following detailed description when read with the accompanying drawings in which:

FIG. 1 is a block illustrating an embodiment of an apparatus for mapping resources amongst virtual machine(s) in accordance with a known technique;

FIG. 2 is a flowchart illustrating an embodiment of a technique for mapping resources amongst virtual machine(s) in accordance with the disclosed subject matter; and

FIG. 3 is a block diagram illustrating an embodiment of an apparatus that allows for management of virtual machine(s) using information regarding the activity of the virtual machine(s) in accordance with the disclosed subject matter.

DETAILED DESCRIPTION

In the following detailed description, numerous details are set forth in order to provide a thorough understanding of the present disclosed subject matter. However, it will be understood by those skilled in the art that the disclosed subject matter may be practiced without these specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail so as to not obscure the disclosed subject matter.

FIG. 2 is a flowchart illustrating an embodiment of a technique for mapping resources amongst virtual machine(s) in accordance with the disclosed subject matter. Block 210 illustrates that a virtual machine manger (VMM) or some other device may detect that a first virtual machine (VM) is attempting to send data to a second VM. It is contemplated that, in one embodiment, the first VM may be expressly attempting to communicate with the second VM, such as, for example, transmitting an internet packet to the second VM. In another embodiment, the communication between the first and second VMs may be implicit. In one example, the output of a device on the first VM

may be virtually coupled with the input of a device on the second VM. In instance of this is where the headphone port of the first VM's virtual sound device is configured to be coupled with the microphone port of the second VM's sound card. However, these are merely a few examples to which the disclosed subject matter is not limited.

Block 220 illustrates that the VMM may map a virtual memory element of the first VM to a shared physical memory element. In one embodiment, the VMM may accomplish this when a virtual device is mapped to a physical device. It is contemplated that any virtual memory elements of the virtual device may be mapped to the shared physical memory element. In one embodiment, the shared physical memory element may be part of a physical device that substantially corresponds to the virtual device, or the shared physical memory element may not correspond to the virtual device. It is contemplated that, in one embodiment, the shared physical memory element may be system memory or a direct memory access (DMA) capable element.

In one embodiment, the VMM may statically map the virtual memory element. Upon starting the first or the second VM, the VMM may detect that the first VM is configured to transmit data to the second VM. In one embodiment, a user may have explicitly instructed the VMM to virtually wire the two VMs together. In another embodiment, the two VMs may be implicitly virtually wired together, such as for example, a private network that only includes the VMs. In the statically mapped embodiment, the virtual memory element of the first VM may be statically mapped to the shared physical memory element. Likewise, the virtual memory element of the second VM may also be mapped to the shared physical memory element. In a specific embodiment, the virtual memory elements of the VMs may be parts of respective virtual

network interfaces. In this specific embodiment, the virtual transmit buffer of the first VM may be statically mapped to a direct memory access (DMA) buffer. The virtual receive buffer of the second VM may also be statically mapped to the DMA buffer. It is contemplated that a second embodiment may occur that maps the virtual receive buffer of the first VM to another DMA buffer, and statically maps the transmit buffer of the second VM to the new DMA buffer. In this embodiment, the two VMs may be statically coupled in a matter that replicates a virtual cross-talk network connection.

In one embodiment, the VMM may dynamically remap the virtual memory element. If the VMM detects that the first VM is not attempting to send data to the second VM, the virtual memory element may be mapped to a physical device that substantially corresponds to the virtual device. In a specific illustrative embodiment, if the first VM is using a network interface to communicate with an outside device, the VM's virtual network interface may be mapped to the physical network interface. However, if the VMM detects that first VM is attempting to communicate with the second VM, the virtual memory element may be dynamically remapped to the shared physical memory element. In a specific illustrative embodiment, if the first VM is using a network interface to communicate with the second VM, the first VM's virtual network interface, specifically the transmit and receive buffers of the virtual network interface, may be dynamically remapped from the physical network interface to respective shared physical memory elements.

Block 230 illustrates that the VMM, or other device, may map a virtual memory element of the second VM to the shared physical memory element. As described above, in one embodiment, the VMM may accomplish this when a virtual device is mapped to a

physical device. It is contemplated that any virtual memory elements of the virtual device may be mapped to the shared physical memory element. In one embodiment, the shared physical memory element may be part of a physical device that substantially corresponds to the virtual device, or the shared physical memory element may not correspond to the virtual device. It is contemplated that, in one embodiment, the shared physical memory element may be system memory or a direct memory access (DMA) capable element.

In one embodiment, the VMM may dynamically remap the virtual memory element. If the VMM detects that the first VM is not attempting to send data to the second VM, the virtual memory element may be mapped to a physical device that substantially corresponds to the virtual device. However, if the VMM detects that first VM is attempting to communicate with the second VM, the virtual memory elements of the first and second VMs may be dynamically remapped to the shared physical memory element.

In a specific illustrative embodiment, if the first VM is using a network interface to communicate with the second VM, the first VM's virtual transmit buffers of the virtual network interface may be dynamically remapped from the physical network interface to a first shared physical memory element. The second VM's virtual receive buffers of the second VM's virtual network interface may also be dynamically remapped to the first shared memory element. In order to enable two-way communication, the first VM's virtual receive buffer may be remapped to a second shared physical memory element, and the second VM's virtual transmit buffers may also be remapped to the second shared physical memory element.

Block 240 illustrates that once the two VMs are coupled via the shared physical memory element, the VMM may wait to detect that information has been placed in the shared physical memory element. In one embodiment, the VMM may monitor the shared memory element. In another embodiment, the VMM may monitor the activity of the first VM. In yet another embodiment he first VM may signal the VMM that the data has been placed in the shared memory element. However, these are merely a few non-limiting examples to which the disclosed subject matter is not limited.

Block 250 illustrates that the VMM, or as previously stated another device, may inform the second VM that data has been received in the VM's virtual memory element. The data may then be processed by the second VM. In one embodiment, the second VM may automatically detect that data has been received by its virtual memory element.

In a specific embodiment, of the technique illustrated by FIG. 2, two VMs may desire to communicate with one another using their respective Ethernet adapters. Other devices are within the scope of the disclosed subject matter; this however is a specific example. The VMM may have mapped the transmit buffer of the first VM's virtual Ethernet adapter to a shared physical direct memory access (DMA) buffer. The VMM may have also mapped the receive buffer of the second VM's virtual Ethernet adapter to the same shared physical direct memory access (DMA) buffer.

The guest driver in the first VM may place a packet to send in the shared DMA buffer, which is pointed to by the Ethernet adapter's transmit descriptor tail register (TDT). The guest driver may move the TDT to reflect that data has been written. The VMM may detect that the virtual TDT of the first VM has been altered. The VMM may move the receive descriptor head register (RDH) in the second VM's Ethernet adapter by

the same number of entries as the TDT moved by the first VM. The VMM may mark the transmission as being complete, by updating the status bits of the receive descriptor for the second VM. The VMM may also inject any receive interrupts into the second VM to notify the guest driver that data has been received. Once the second VM's guest driver has read the data, by moving the receive descriptor tail register (RDT) past the new data entries, the VMM may inform the first VM, via interrupts and alteration of the virtual Ethernet register, that the transmission is complete. In this example, the shared physical memory element allows the two VMs to communicate as if they were physically connected via a cross-over Ethernet cable.

In one embodiment, the virtual devices of the two VMs may be substantially identical. In other embodiments, the two virtual devices may be of the same category (*e.g.* sound card, network adapter), but of different models or manufacturers. It is also contemplated that while one-way communication was illustrated by FIG. 2, bi-directional and one-to-many communication is within the scope of the disclosed matter. In one embodiment, bi-directional communication may utilize two shared memory elements, one for each direction. In another embodiment, bi-directional communication may utilize a single shared memory element. In one embodiment, the shared memory element may include many elements, and may not be contiguous.

FIG. 3 is a block diagram illustrating an embodiment of an apparatus 301 that allows for management of virtual machine(s) using information regarding the activity of the virtual machine(s) in accordance with the disclosed subject matter. Virtual machine manager 380 may include a cross-talk detector 385 and a dynamic memory remapper

387. In one embodiment, the VMM may be capable of performing a technique similar to the one described above and illustrated by FIG. 2.

The cross-talk detector 385 may be capable of detecting when a first VM 311 is attempting to transmit data to a second VM 312. The dynamic memory remapper 387 may be capable of dynamically remapping the first virtual memory 321, which is part of the first virtual device 331, to shared physical memory element 350. The dynamic memory remapper may also be capable of remapping the second virtual memory 322, which is part of the second virtual device 332, to the shared physical memory element. The dynamic memory remapper may also be capable of mapping the first virtual memory element to the physical memory element 340 of the physical device 390. In one embodiment, the dynamic remapper may map the virtual memory element to the shared physical memory element or the physical memory element of the physical device based, at least in part, upon whether the cross-talk detector determines that the first VM is attempting to transmit data to the second VM or a device outside the apparatus 301.

In one embodiment, the cross-talk detector 385 may be capable of performing the technique illustrated by block 210 of Fig. 2. In one embodiment, the dynamic remapper may be capable of performing the technique illustrated by block 220 & 230 of Fig. 2. In one embodiment, the VMM 380 may be capable of performing the technique illustrated by blocks 240 & 250 of Fig. 2. In another embodiment, the technique illustrated by blocks 240 & 250 of Fig. 2 may instead be performed by the cross-talk detector.

The techniques described herein are not limited to any particular hardware or software configuration; they may find applicability in any computing or processing environment. The techniques may be implemented in hardware, software, firmware or a

combination thereof. The techniques may be implemented in programs executing on programmable machines such as mobile or stationary computers, personal digital assistants, and similar devices that each include a processor, a storage medium readable or accessible by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices. Program code is applied to the data entered using the input device to perform the functions described and to generate output information. The output information may be applied to one or more output devices.

Each program may be implemented in a high level procedural or object oriented programming language to communicate with a processing system. However, programs may be implemented in assembly or machine language, if desired. In any case, the language may be compiled or interpreted.

Each such program may be stored on a storage medium or device, *e.g.* compact disk read only memory (CD-ROM), digital versatile disk (DVD), hard disk, firmware, non-volatile memory, magnetic disk or similar medium or device, that is readable by a general or special purpose programmable machine for configuring and operating the machine when the storage medium or device is read by the computer to perform the procedures described herein. The system may also be considered to be implemented as a machine-readable or accessible storage medium, configured with a program, where the storage medium so configured causes a machine to operate in a specific manner. Other embodiments are within the scope of the following claims.

While certain features of the disclosed subject matter have been illustrated and described herein, many modifications, substitutions, changes, and equivalents will now

occur to those skilled in the art. It is, therefore, to be understood that the appended claims are intended to cover all such modifications and changes that fall within the true spirit of the disclosed subject matter.